Lecture 3: Modern Methods of Statistical Learning sf2935 : Neural Networks: Learning Timo Koski

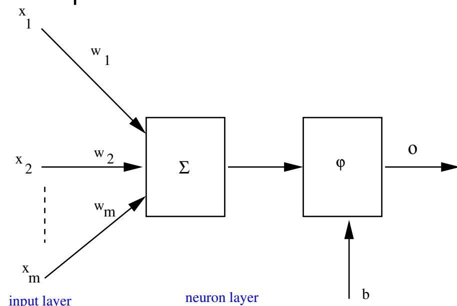
TK

2018-09-03

- Forward neural networks with multiple layers
- The supervised learning task for neural networks
- Backpropagating
- Neural networks and statistics
- Bias-variance trade-off

Neuron: Input Layer and One Neuron Layer

$\mathbf{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$ = input space, an inner product space of finite dimension $= m$. $\varphi(x)$= activation function or activator, examples later.



$\mathbf{w} = (w_1, \ldots, w_m)$
$o = \varphi\left(\sum_{i=1}^{m} w_i x_i + b_j\right)$
$\mathbf{w}$ is called the weight vector and b is a bias.

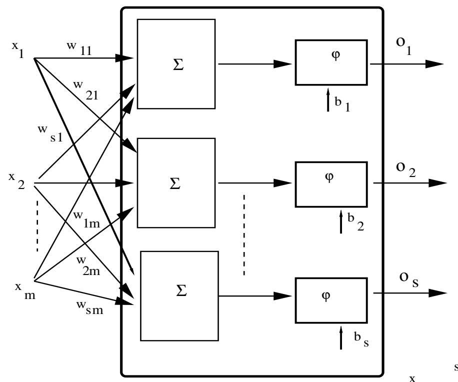
$\varphi(\cdot)$ is a sigmoidal function, if it is nonconstant, bounded, and monotonically-increasing continuous function.

Example

The logistic function:

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

Neural Network: Input Layer & a Layer of s Parallell Neurons



$$\mathbf{w}_j = (w_{j1}, \ldots, w_{jm}) \; j = 1, \ldots, s$$
$$o_j = \varphi\left(\sum_{i=1}^{m} w_{ji} x_i + b_j\right), \; j = 1, \ldots, s$$

Composition of Functions

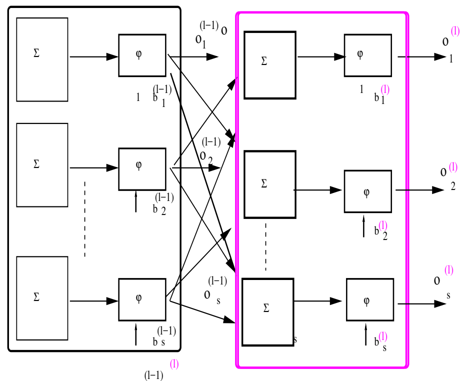
$f(x)$ and $g(x)$ are two functions such that the domain of f includes the range of g, the composition $(fog)(x)$ of $f(x)$ and $g(x)$ is defined as

$$(fog)(x) \stackrel{\text{def}}{=} f(g(x)).$$

Chain rule of differentiation:

$$\frac{d}{dx}(fog)(x) = f^{'}(g(x)) \cdot g^{'}(x).$$

Composition of Layers



Output of layer l $o_j^{(l)} = \varphi\left(\sum_{i=1}^{s} w_{ji}^{(l)} o_i^{(l-1)} + b_j^{(l)}\right), j = 1, \ldots, s$

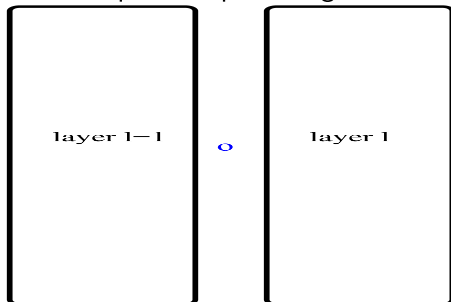
$b_j^{(l)}$ = bias of the jth neuron in the lth layer

$w_{ji}^{(l)}$ = the weight of the ith neuron in the lth layer to the jth neuron in the lth layer.

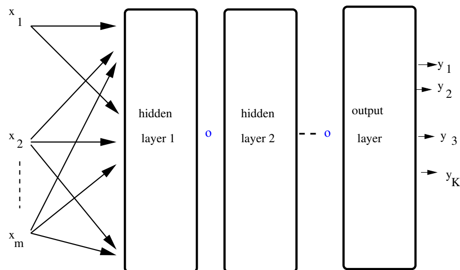
$o_i^{(l-1)}$ = the output of the ith neuron in the $l-1$th layer.

Composition of Layers

Let us depict the preceding as



Neural Network with L layers (composed), layer number L is the output layer



$y_j = \varphi\left(\sum_{i=1}^{m_{L-1}} w_{ji}^{(L)} o_i^{(L-1)} + b_j^{(L)}\right)$, $j = 1, \ldots, K$, the outputs of the neural network. The layers preceding the output layer are called hidden layers, as we do not register their outputs. There is **no feedback**, all computation is feedforward from input layer to output layer.

SUPERVISED LEARNING

t is the target of the input $\mathbf{x}$.
Training set: $\mathcal{S} = \left\{ \left(\mathbf{x}^{(r)}, t^{(r)} \right) \right\}_{r=1}^{n}$ consists of the n data pairs of inputs labelled with the targets $t^{(r)}$.

Examples of target domains

- Supervised classification with K classes: target space consists of vectors t with $K-1$ zeros and a single one. Hence

$$t = (0, 0, \ldots, \underbrace{1}_{\text{k:th position}}, 0, \ldots, 0)$$

  indicates class k,
- Estimation of an unknown real valued function f from n values. $K = 1$,

$$t^{(r)} = f\left(\mathbf{x}^{(r)}\right), r = 1, 2, \ldots, n,$$

  the target space $= \mathbb{R}$.

Quadratic cost function

Since the outputs of the neural network are

$$y_j = \varphi\left(\sum_{i=1}^{m_{L-1}} w_{ji}^{(L)} o_i^{(L-1)} + b_j^{(L)}\right), j = 1, \ldots, K$$

we can write them as (composed) functions of the inputs

$$y_j = y_j\left(\mathbf{x}\right).$$

We set

$$C\left(\mathbf{w}, \mathbf{b}\right) \stackrel{\text{def}}{=} \frac{1}{2n} \sum_{r=1}^{n} \sum_{i=1}^{K} (y_i(\mathbf{x}^{(r)}) - t_i^{(r)})^2$$

$\mathbf{w}$ denotes the collection of all the weights in the network, $\mathbf{b}$ all the biases, n is the total number of training inputs, $\mathbf{y} = \left(y_1(\mathbf{x}^{(r)}), \ldots, y_L(\mathbf{x}^{(r)})\right)$ is the vector of outputs from the network when $\mathbf{x}^{(r)}$ is input, and the sum is over all training inputs.

Learning algorithms which can automatically tune the weights and biases of a network of artificial neurons so that

$$C(\mathbf{w}, \mathbf{b}) = \frac{1}{2n} \sum_{r=1}^{n} \sum_{i=1}^{K} (y_i(\mathbf{x}^{(r)}) - t_i^{(r)})^2$$

is minimized, where $\left\{ \left(\mathbf{x}^{(r)}, t^{(r)} \right) \right\}_{r=1}^{n}$ is the training set. This is called Minimization of the Mean Squared Error (MSE).

KTH Matematik

Gradient Descent

Gradient descent is an iterative algorithm for finding the minimum of a differentiable function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point. Or, in the neighborhood of a point, a function decreases fastest if one goes from the point in the direction of the negative gradient of the function.

KTH Matematik

Gradient Descent

We write $C = C(\mathbf{w}, \mathbf{b})$. The gradient descent method involves in this cae calculating the gradient of C with respect to the weights of the network.

We have a current value of a weight w_{ij}^{old} (dropping the layer index (l) for simplicity of writing). We update the w_{ij}^{old} using gradient descent,

$$w_{ij}^{old} \to w_{ij}^{old} + \Delta w_{ij}.$$

The change in weight, which is added to the old weight, is equal to the product of the learning rate η and the gradient, multiplied by -1:

$$\Delta w_{ij} = -\eta \frac{\partial C}{\partial w_{ij}}.$$

The -1 is required in order to update in the direction of a minimum, not a maximum, of the cost function.

Backpropagation means computing the partial derivatives $\frac{\partial C}{\partial w_{ij}^{(l)}}$ and $\frac{\partial C}{\partial b_j^l}$ in a recursive manner from the output layer to the input

KTH Matematik

Backpropagation actually allows us to compute the partial derivatives $\frac{\partial C}{\partial w_{ij}}$ and $\frac{\partial C}{\partial b_j}$ for a single training example. We can then recover $\frac{\partial C}{\partial w_{ij}}$ and $\frac{\partial C}{\partial b_j}$ by averaging over training examples.
In fact,

- we fix a single training example $\mathbf{x}$. We remove it from notation as it is a notational nuisance.
- $$C = \frac{1}{2} \sum_{i=1}^{K} (t_i - y_i(\mathbf{x}))^2.$$

  The factor $\frac{1}{2}$ cancels the exponent, when differentiating.

For simplicity of writing we drop also the layer index (l) in

$$o_j = \varphi\left(\sum_{i=1}^{m} w_{ji} o_i + b_j\right).$$

We set

$$\text{net}_j \stackrel{\text{def}}{=} \sum_{i=1}^{m} w_{ji} o_i + b_j,$$

so that

$$o_j = \varphi(\text{net}_j).$$

The input net_j to an activator φ is the weighted sum of outputs o_k of neurons in the previous layer.
If the neuron is in the first layer after the input layer, the o_k of the input layer are simply the inputs x_k to the network. The number of input units to the neuron is m. The variable w_{ji} denotes the weight between neuron i at previous layer and neuron j at the layer above.

Chain Rule

Calculation of the partial derivative of the error C with respect to a weight w_{ji} is done using the chain rule twice:

$$\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial o_j}\frac{\partial o_j}{\partial w_{ji}} = \frac{\partial C}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ji}} \tag{1}$$

The logistic function

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

has the derivative:

$$\frac{d\varphi(z)}{dz} = \varphi(z)(1 - \varphi(z))$$

Assume that the logistic function is used, then the middle factor in the right hand side of (1) is

$$\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial}{\partial \text{net}_j} \varphi(\text{net}_j)$$

$$= \varphi(\text{net}_j)(1 - \varphi(\text{net}_j)).$$

KTH Matematik

$\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j}\frac{\partial \text{net}_j}{\partial w_{ji}}$: Output Layer

The first factor $\frac{\partial C}{\partial o_j}$ in the right hand side of (1), the derivative of the output of $y_j = o_j$ with respect to its input is simple, if the neuron is in the output layer, as then $y_j = o_j$ and

$$\frac{\partial C}{\partial o_j} = \frac{\partial C}{\partial y_j}$$

$$= \frac{\partial}{\partial y_j}\frac{1}{2}\sum_{i=1}^{K}(t_i - y_i)^2 = y_j - t_j = o_j - t_j.$$

$\frac{\partial \text{net}_j}{\partial w_{ji}}$

In the last factor of in the right hand side of (1), only one term in net_j depends on w_{ji}. Therefore

$$\frac{\partial \text{net}_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left(\sum_{k=1}^{n} w_{jk} o_k \right) = \frac{\partial}{\partial w_{ji}} w_{ji} o_i = o_i.$$

KTH Matematik

Output Layer: Put it together

For the weights in the output layer we have now obtained in the above for (1):

$$\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= (o_j - t_j) \cdot oj(1 - o_j) \cdot o_i \cdot$$

$\frac{\partial C}{\partial w_{ji}}$: Other layers

To calcuate here, we first introduce an intermediate quantity, δ_j^l, which we call **the error in the jth neuron (in the lth layer)**. We drop l for simplicity, and define

$$\delta_j \stackrel{\text{def}}{=} \frac{\partial C}{\partial \text{net}_j} \tag{2}$$

We have by the chain rule of calculus

$$\frac{\partial C}{\partial w_{ji}} = \frac{\partial C}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ji}} = o_i \delta_j. \tag{3}$$

and

$$\delta_j = \frac{\partial C}{\partial \text{net}_j} = \frac{\partial C}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \varphi^{'}(\text{net}_j) \frac{\partial C}{\partial o_j}. \tag{4}$$

KTH Matematik

$\frac{\partial C}{\partial w_{ji}}$: the difficulties at earlier layers

In the calculations to follow the difficulty is that one must make very clear what is fixed. Much (?) of the machine learning literature does not do this with backpropagation and is well confused or difficult to understand.
When computing $\frac{\partial C}{\partial w_{ji}}$ we regard C as function of all the weights. Changes in w_{ji} affect the net_j and o_j and all neurons taking input from neuron j including some associated os.

$\frac{\partial C}{\partial o_j}$: the difficulties at earlier layers

When we take partial derivatives w.r.t. to a net or an output of a neuron, we allow all other signals in the network which depend on the net or the output of a neuron to follow their usual dependence. Thus all weights and all nets (and hence corresponding o outputs) of neurons **in the same and earlier layers are kept fixed**.

We evaluate $\delta_j = \frac{\partial C}{\partial \text{net}_j}$ by noting that net_j affects the neuron outputs through o_j, and **this only acts through connections to other neurons's outputs at layers closer to the network output layer**.

$\frac{\partial C}{\partial o_j}$

By (4) we have

$$\delta_j = \varphi^{'}(\text{net}_j)\frac{\partial C}{\partial o_j}.$$

Let now o_k be a neuron at the next layer ($l+1$, this is closer to the output layer) and taking input from neuron net_j at layer l. We indicate this with

$$k : j \to k$$

KTH Matematik

Then we take the total derivative w.r.t. to o_j to obtain

$$\frac{\partial C}{\partial o_j} = \sum_{k:j \to k} \frac{\partial C}{\partial \mathrm{net}_k} \frac{\partial \mathrm{net}_k}{\partial o_j}$$

and by chain rule

$$= \sum_{k:j \to k} \frac{\partial C}{\partial o_k} \frac{\partial o_k}{\mathrm{net}_j} w_{kj}$$

and by the middle equality in (4)

$$= \sum_{k:j \to k} \delta_k w_{kj}$$

KTH VETENSKAP OCH KONST

KTH Matematik

$$\frac{\partial C}{\partial o_j} = \sum_{k:j \to k} \delta_k w_{kj}$$

and

$$\delta_j = \varphi^{'}(\text{net}_j) \sum_{k:j \to k} \delta_k w_{kj}$$

Therefore, the derivative with respect to o_j can be calculated if all the derivatives with respect to the outputs o_k of the next layer - the one closer to the output neuron - are known.

KTH Matematik

Put it all together:

$$\frac{\partial C}{\partial w_{ij}} = o_i \delta_j$$

where

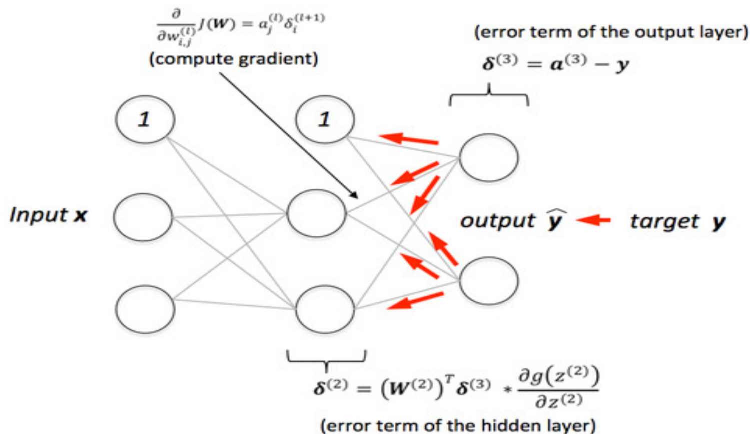
$$\delta_j = \frac{\partial C}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{if } j \text{ is an output neuron,} \\ \left(\sum_{k:j \to k} \delta_k w_{kj}\right) o_j (1 - o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

Backpropagation:

The error vectors δ_j are computed backward, starting from the final layer. The backward movement is a consequence of the fact that the cost is a function of outputs from the network.

- a **forward pass** to calculate the outputs from the inputs
- a **backward pass** to calculate δ_j

Backpropagation:



KTH VETENSKAP OCH KONST
KTH Matematik

Who discovered backpropagation?

Neural Network: Universal Approximator (Cybenko & Hornik)

Let $\varphi(\cdot)$ be a sigmoidal function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$.
Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$ such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi \left(\sum_{k=1}^{m} w_{ik} x_k + b_i \right)$$

as an approximate realization of the function f, where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

Neural Networks and Statistics

Let us write

$$C(\mathbf{w},\mathbf{b}) = \frac{1}{2n}\sum_{r=1}^{n}\sum_{i=1}^{K}(y_i(\mathbf{x}^{(r)}) - t_i^{(r)})^2 = \frac{1}{2n}\sum_{r=1}^{n}\sum_{i=1}^{K}(t_i^{(r)} - y_i(\mathbf{x}^{(r)}))^2.$$

Set $(\mathbf{t}^{(\mathbf{r})} = (t_1^{(r)}, \ldots, t_K^{(r)}$. Let us assume that the pairs $(\mathbf{t}^{(\mathbf{r})}, \mathbf{x}^{(r)})$, $r = 1, \ldots n$ are independent samples with the probability distribution $P(\mathbf{T}, \mathbf{X})$. Then the Law of Large Numbers gives

$$C(\mathbf{w},\mathbf{b}) \approx \frac{1}{2}E\left[||\mathbf{T} - y(\mathbf{X})||^2\right]$$

for large n, where

$$||\mathbf{t} - y||^2 = \sum_{i=1}^{K}(t_i - y_i)^2$$

$$C(\mathbf{w}, \mathbf{b}) \approx \frac{1}{2} E\left[||\mathbf{T} - y(\mathbf{X})||^2\right]$$

and probability theory tells that $E\left[||\mathbf{T} - y(\mathbf{X})||^2\right]$ is minimized by

$$y^*(\mathbf{X}) = E\left[\mathbf{T} \mid \mathbf{X}\right]$$

Hence we can say that a neural network after learning (with mean squred error cost criterion) approximates the conditional expectation of the target variable given the data.

For example, in a two-category classification problem, if the target t is coded as 1, if the object is from class 1 and -1 if it is from class 2. Then,

$$E\left[\mathbf{T} \mid \mathbf{X}=\mathbf{x}\right] = +1 \cdot P(\omega_1 \mid \mathbf{x}) + (-1) \cdot P(\omega_2 \mid \mathbf{x})$$

and the neural network estimates the following (well-known) discriminant function:

$$P(\omega_1 \mid \mathbf{x}) - P(\omega_2 \mid \mathbf{x}).$$

- M. D. Richard and R. Lippmann: Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural Computation*, vol. 3, pp. 461– 483, 1991.

Bias -Variance Trade-Off

Suppose that we have a training set consisting of a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and real values t_i associated with each point x_i. We assume that there is a functional, but noisy relation

$$t_i = f(\mathbf{x}_i) + \epsilon,$$

where the noise, ϵ, has zero mean and variance σ^2 .
We want to find a function $\hat{f}(\mathbf{x})$, that approximates the true function $t = f(\mathbf{x})$ as well as possible, by means of some learning algorithm. We make "as well as possible" precise by measuring the MSE between y and $\hat{f}(\mathbf{x})$ which we want to be minimal both for $\mathbf{x}_1, \ldots, \mathbf{x}_n$ AND for points outside of our sample. Of course, we cannot hope to do so perfectly, since the y_i contain noise ϵ. This means we must be prepared to accept an irreducible error in any function we come up with.

Bias -Variance Decomposition

Finding an $\hat{f}$ that generalizes to points outside of the training set can be done with any of the countless algorithms used for supervised learning. It turns out that whichever function $\hat{f}$ we select, we can decompose its expected error on an unseen sample $\mathbf{x}$ as follows:

$$\mathrm{E}\left[\left(t - \hat{f}(\mathbf{x})\right)^2\right] = \mathrm{Bias}\left[\hat{f}(\mathbf{x})\right]^2 + \mathrm{Var}\left[\hat{f}(\mathbf{x})\right] + \sigma^2 \tag{5}$$

$$\tag{6}$$

Where:

$$\mathrm{Bias}\left[\hat{f}(\mathbf{x})\right] = \mathrm{E}\left[\hat{f}(\mathbf{x})\right] - f(\mathbf{x}) \tag{7}$$

and

$$\mathrm{Var}\left[\hat{f}(\mathbf{x})\right] = \mathrm{E}\left[\left(\hat{f}(\mathbf{x}) - \mathrm{E}[\hat{f}(\mathbf{x})]\right)^2\right] \tag{8}$$

The expectation ranges over different choices of the training set $\mathbf{x}_1, \ldots, \mathbf{x}_n, t_1, \ldots, y_n$ all sampled from the same distribution. The three terms represent:

- the square of the bias of the learning method, which can be thought of the error caused by the simplifying assumptions built into the method. E.g., when approximating a non-linear function $f(\mathbf{x})$ using a learning method for linear models, there will be error in the estimates $\hat{f}(\mathbf{x})$ due to this assumption;
- the variance of the learning method, or, intuitively, how much the learning method $\hat{f}(\mathbf{x})$ will move around its mean;
- the irreducible error σ^2. Since all three terms are non-negative, this forms a lower bound on the expected error on unseen samples.

Bias -Variance Trade-Off

Optimum Model Complexity

Total Error

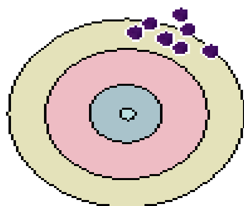
Error

Variance

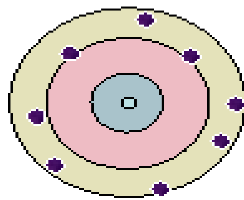
Bias^2

Model Complexity

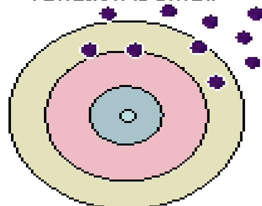
Bias -Variance Trade-Off



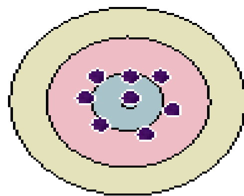
Bias is large
variation is small



Bias is small
variation is large



Bias is large
variation is large



Bias is small
variation is small

Accuracy versus Quality of an Estimator Using Bias and Variation as Measurable Quantities Respectively

Bias -Variance Trade-Off

- *If the model is too simple, the solution it yields is biased and does not fit the data.*
- *If the model is too complex, it is too sensitive to small variations in data*

KTH Matematik

Ideally, the optimal model that minimizes the overall MSE given by $E[\mathbf{T}|\mathbf{X}]$, which leaves the minimum MSE to be the intrinsic error σ^2. Because of the randomness of the limited training set, the estimate ANN is also a random variable which will hardly be the $E[\mathbf{T}|\mathbf{X}]$ for a given training set. The bias and best possible function variance terms hence provide useful information on how the estimation differs from the desired function.

Bias -Variance Trade-Off

The model bias measures the extent to which the average of the estimation function over all possible data sets with the same size differs from the desired function. The model variance, on the other hand, measures the sensitivity of the estimation function to the training data set. Although it is desirable to have both low bias and low variance, we can not reduce both at the same time for a given data set because these goals are conflicting.